Final Project Progress

Emma Chen, Xavier Routh, William Yeh

Please remember to prepare **3 to 5 pages of slides** for these meetings.

This will help you better introduce the paper you are working on, propose the new methodology and describe the progress achieved by your team s

November 13-20: **10% of the grade**

Each team will sign up for a 15 minute progress meeting with the TA. Before the meeting, the students should write the progress report, 1-page PDF and email it to the TA. By this stage you should have a *partial working code for a subset of the proposed functionality* that has been tested on test cases you wrote. The functionality can be small, but it should be solid and non-trivial (original to your project). Describe what you have accomplished, including any relevant preliminary results for programs that work. Note that 30% of the overall project grade is reserved for your progress accomplished during the first month including your project proposal.

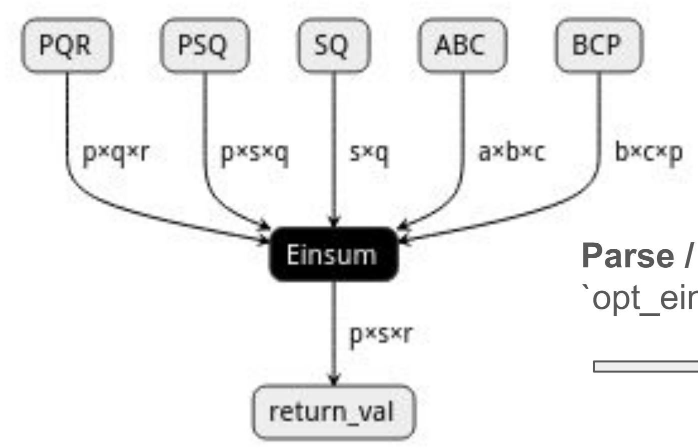
Einsum Introduction

- Einsum expressions lowered into a tree of binary operations between input tensors
- Einsum contraction paths (binary trees) in terms of FLOPs and Memory have good solutions already
- Large einsum expressions can benefit from global optimization to optimize for data layout.

- Given a contraction tree (that is already optimized for FLOPs and Memory):
- **Einsum Tree IR** is used to optimize for data layout, and to target fast primitives (transpose, GEMM, packed GEMM)

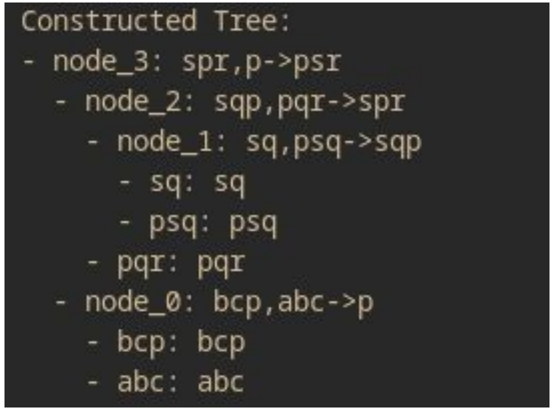
Progress 1

ONNX Expression:
pqr,psq,sq,abc,bcp->psr



Parse / Find contraction path:
`opt_einsum.contract_path("pqr, …->")

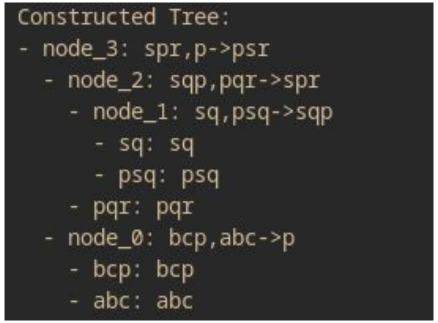
Einsum Tree IR



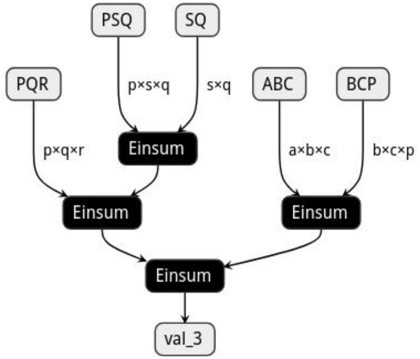
```
Constructed Tree:
- node_3: spr,p->psr
  - node_2: sqp,pqr->spr
    - node_1: sq,psq->sqp
      - sq: sq
      - psq: psq
    - pqr: pqr
  - node_0: bcp,abc->p
    - bcp: bcp
    - abc: abc
```

Progress 2

Unoptimized Einsum Tree IR

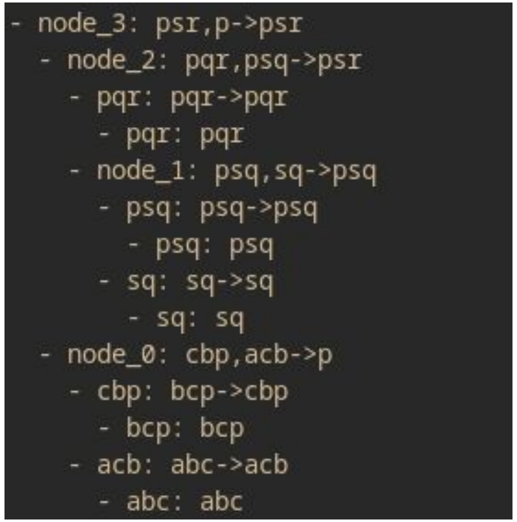


```
Constructed Tree:
- node_3: spr,p->psr
  - node_2: sqp,pqr->spr
    - node_1: sq,psq->sqp
      - sq: sq
      - psq: psq
    - pqr: pqr
  - node_0: bcp,abc->p
    - bcp: bcp
    - abc: abc
```

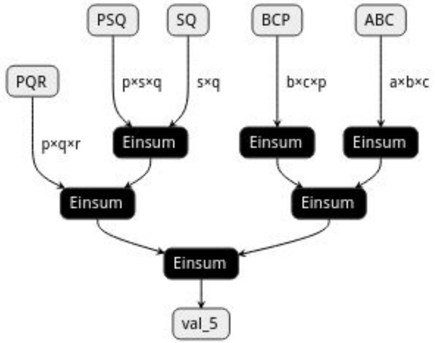


Optimization from Einsum Tree Paper

Optimized Einsum Tree IR



```
- node_3: psr,p->psr
  - node_2: pqr,psq->psr
    - pqr: pqr->pqr
      - pqr: pqr
    - node_1: psq,sq->psq
      - psq: psq->psq
        - psq: psq
      - sq: sq->sq
        - sq: sq
  - node_0: cbp,acb->p
    - cbp: bcp->cbp
      - bcp: bcp
    - acb: abc->acb
      - abc: abc
```



Results

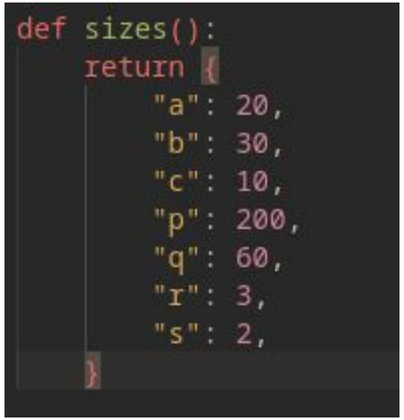
Setup:
Personal computer CPU
(ONNX RT provider)
20 runs

Numpy Reference:
np.einsum(expr)

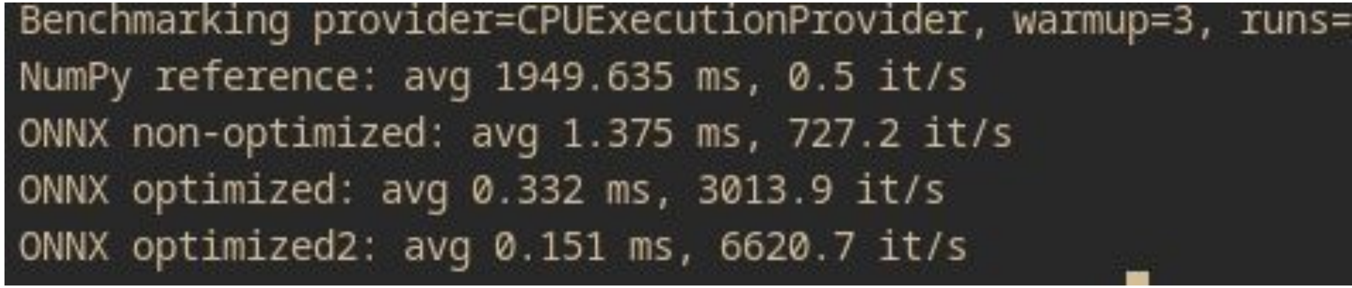
ONNX non-optimized:
Single onnx einsum node

ONNX optimized:
Original contraction tree

ONNX optimized2:
Einsum Tree paper optimized contraction tree



```
def sizes():
    return {
        "a": 20,
        "b": 30,
        "c": 10,
        "p": 200,
        "q": 60,
        "r": 3,
        "s": 2,
    }
```



```
Benchmarking provider=CPUExecutionProvider, warmup=3, runs=
NumPy reference: avg 1949.635 ms, 0.5 it/s
ONNX non-optimized: avg 1.375 ms, 727.2 it/s
ONNX optimized: avg 0.332 ms, 3013.9 it/s
ONNX optimized2: avg 0.151 ms, 6620.7 it/s
```

Next Steps

- Benchmark on GPU
- Codegen (lowering contraction path to loops + 3 primitives):
    - **Problem:** Attempts to build into ONNX fail, as onnx doesn’t support packed GEMM. No data layout definitions in ONNX.
    - **Current Solution**: Leave to ONNX RT.
    - **Future Solution**: Build codegen algorithm into XLA frontend, as it rewrites einsum expressions (stableHLO). See if we can enforce intermediate data layouts this way.
- Evaluate impact of original contraction path